Національний технічний університет України

**“Київський політехнічний інститут імені Ігоря Сікорського”**

**Кафедра автоматизації проектування  
енергетичних процесів і систем**

***КУРСОВА РОБОТА***

**з дисципліни: «Об’єктно-орієнтоване програмування»**

**тема:**

**«Створення моделей об’єктів реального світу на мові C++»**

|  |  |
| --- | --- |
| **Керівник – Карпенко С.Г.** | **Виконала Малько Т.В.** |
| **Допущено до захисту** | **Студентка 2-го курсу** |
| **\_\_ \_\_\_\_\_\_\_\_ 2020 р.** | **Групи ТР-91** |
| **Захищено з оцінкою** | **залікова книжка № ТР-91109** |
| **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |  |

***2020***

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Кафедра автоматизації проектування  
енергетичних процесів і систем

**КУРСОВА РОБОТА**

з дисципліни «Об’єктно-орієнтоване програмування»

(назва дисципліни)

на тему:

«Створення моделей об’єктів реального світу на мові C++»

Студента групи **ТР-91**

спеціальність **122 «Комп’ютерні науки»**

освітня програма Комп’ютерний моніторинг та геометричне моделювання процесів та систем

Малько Т.В.

(прізвище та ініціали)

Керівник доцент, канд. фіз.-мат. наук Карпенко С.Г.

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_Оцінка: ECTS \_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_ доц., к. ф.-м. н. Карпенко С.Г.

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_ доц., к.е.н. Левченко Л.О.

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_ доц., к.т.н. Шаповалова С.І.

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали

Київ - 2020 рік

**Національний технічний університет України**

**“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет ТЕПЛОЕНЕРГЕТИЧНИЙ ФАКУЛЬТЕТ

( повна назва)

Кафедра автоматизації проектування енергетичних процесів та систем

(повна назва)

Освітньо-кваліфікаційний рівень бакалавр

спеціальність **122 «Комп’ютерні науки»**

освітня програма **Комп’ютерний моніторинг та  
геометричне моделювання процесів та систем**

ЗАВДАННЯ

**НА КУРСОВУ РОБОТУ СТУДЕНТУ**

Малько Тамарі Вікторівні

(прізвище, ім’я, по батькові)

1. Тема роботи «Створення моделей об’єктів реального світу на мові C++»

керівник курсової роботи –

Карпенко Станіслав Григорович, канд. фіз.-мат. наук, доцент

( прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

2. Строк подання студентом роботи – 19 грудня 2020 р.

3. Початкові дані до проекту (роботи): мова C++, модель об’єкту реального світу.

4. Зміст пояснювальної записки курсової роботи (перелік питань, які потрібно розробити) – Розробити та обґрунтувати ієрархічну модель класів для опису об’єкту реального світу, використовуючи інструментальні можливості мови C++, зокрема успадкування, інкапсуляцію, перевантажені операції, шаблони функцій, запис у двійкові та текстові файли та обробку виключень.

5. Перелік графічного матеріалу –

графічне зображення ієрархічної моделі класів з урахуванням як успадкування, так й включення; відображення значень основних параметрів реальних об’єктів у вигляді таблиці.

6. Дата видачі завдання – 03 листопада 2020 р.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів виконання курсової роботи | Строк виконання етапів роботи | Примітка |
|  | Побудова ієрархічної моделі класів | 17.11.2020 |  |
|  | Розробка та застосування інструментальних засобів обробки даних | 01.12.2020 |  |
|  | Написання розрахунково-пояснювальної записки | 19.12.2020 |  |

**Студент \_\_\_\_\_\_\_\_\_ Малько Т.В.**

( підпис ) (прізвище та ініціали)

**Керівник курсової роботи \_\_\_\_\_\_\_\_\_ Карпенко С.Г.**

( підпис ) (прізвище та ініціали)

**АНОТАЦІЯ**

У даній курсовій роботі була виконана розробка програми на мові С++ з побудовою ієрархії класів яка відображає об’єкти, які можуть бути використані для створення системи магазину. Програмний код написаний на основі парадигм об’єктно орієнтованого програмування.

Проведений аналіз основних особливостей вибраної мови програмування.

**ЗМІСТ**

1. ВСТУП 5
   1. Огляд характеристик і особливостей мови С++ 7
   2. Завдання курсової роботи 8
   3. Ієрархія класів 10
   4. Обмеження полів 13
   5. Поведінка об’єктів 17
   6. Оператори 17

4.2 Пошук 21

4.3 Методи 26

* 1. Виключення, обробка виключних ситуацій 27
  2. Ієрархія виключень 27
  3. Обробка виключних ситуацій 28

Висновки 30

Список використаних джерел 31

Додаток 1 Специфікація 32

Додаток 2 Текст програми 34

Додаток 3 Опис програми 53

**Вступ**

**Об'єктно-орієнтоване програмування** — одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою замість того, щоб розглядати її як набір виконуваних інструкцій. Вони можуть володіти різними властивостями та мати набір дій, які з ними можна проводити.

Основу ООП складають чотири основні концепції: інкапсуляція, успадкування, поліморфізм та абстракція. Однією з переваг ООП є краща модульність програмного забезпечення (тисячу функцій процедурної мови, в ООП можна замінити кількома десятками класів із своїми методами). Попри те, що ця парадигма з'явилась в 1960-тих роках, вона не мала широкого застосування до 1990-тих, коли розвиток комп'ютерів та комп'ютерних мереж дав змогу писати надзвичайно об'ємне і складне програмне забезпечення, що змусило переглянути підходи до написання програм. Сьогодні багато мов програмування або підтримують ООП (PHP, Lua) або ж є цілком об'єктно-орієнтованими (зокрема, Java, C#, C++, Python, Ruby і Objective-C, ActionScript 3, Swift, Vala).

C ++ - компільована, статично типізована мова програмування загального призначення.

Підтримує такі парадигми програмування як процедурне програмування, об'єктно-орієнтоване програмування, узагальнене програмування, забезпечує модульність, роздільну компіляцію, обробку виключень, абстракцію даних, оголошення типів (класів) об'єктів, віртуальні функції. Стандартна бібліотека включає, в тому числі, загальновживані контейнери і алгоритми. C ++ поєднує властивості як високорівневих, так і низькорівневих мов. У порівнянні з його попередником - мовою C, - найбільшу увагу приділено підтримці об'єктно-орієнтованого і узагальненого програмування.

C ++ широко використовується для розробки програмного забезпечення та є одною з найпопулярніших мов програмування. Область її застосування включає створення операційних систем, різноманітних прикладних програм, драйверів пристроїв, додатків для вбудованих систем, високопродуктивних серверів, а також розважальних програм. Існує безліч реалізацій мови C ++, як безкоштовних, так і комерційних і для різних платформ.

Ця мова підтримує успадкування - механізм утворення нових класів на основі використання вже існуючих. При цьому властивості та функціональність батьківського класу переходять до класу нащадка. Якщо один клас є нащадком іншого, то похідний клас має можливість користуватися всіма даними і функціями-членами, визначенимив базовому класі. Тобто клас може просто використовувати всю основну функціональність, що надається батьківським, і додавати тільки те, що потрібно йому для роботи.

Інкапсуляція об'єднує дані і код, який маніпулює ними, а також захищає і те, і інше від зовнішнього втручання або неправильного використання. Разом вони створюють об'єкт, всередині якого можуть бути приватні дані, якими можуть маніпулювати тільки методи об'єкта. Виходячи з цього, у нас з'являється ще одна корисна властивість - приховування реалізації програмних частин.

Поліморфізм – це можливість доповнювати об'єкт функціоналом. Можливість об'єкту знаходитися в різних формах. Класичний поліморфізм - заміщення, перевизначення методів. Ad hoc поліформізм - перевантаження методів, поведінка в залежності від даних. Заміна об'єктів під час виконання програми з однаковими методами через інтерфейс.

Абстракція - в ООП це додання об'єкту характеристик, які чітко виділяє його на тлі інших, визначаючи його концептуальні межі. Абстрагування - В ООП це спосіб виділити набір значущих характеристик об'єкта, виключаючи з розгляду не значимі. Відповідно абстракція це набір таких характеристик.

**Основна частина**

1. Огляд характеристик і особливостей мови С++

Створюючи нові типи для опису предметної області, C ++ допомагає програмісту писати легші для розуміння програми. Класи дозволяють відокремити деталі, що стосуються реалізації нового типу, від визначення інтерфейсу і операцій, що надаються користувачеві.

Ключовою особливістю класу в C ++ є поділ інтерфейсу і реалізації. Інтерфейс являє собою набір операцій (функцій), які виконуються об'єктом; він визначає ім'я функції, яке значення і список параметрів.

Зазвичай користувач не повинен знати про об'єкт нічого, крім його інтерфейсу. Реалізація приховує алгоритми і дані, потрібні об'єкту, і може змінюватися при розвитку об'єкта, ніяк не зачіпаючи інтерфейс.

Для підтримки ініціалізації класу служить конструктор - визначена проектувальником функція (можливо, перевантажена), яка автоматично застосовується до кожного об'єкта класу перед його першим використанням. Парна по відношенню до конструктору функція, деструкція, автоматично застосовується до кожного об'єкта класу після закінчення його використання і призначена для звільнення ресурсів, захваченних або в конструкторі класу, або на протязі його життя.

Перевантажені оператори дозволяють працювати з об’єктами так само просто та зрозуміло як зі стандартними типами даних. Перевантажений оператор оголошується в тілі класу точно так же, як звичайна функція-член, тільки його ім'я складається з ключового слова operator, за яким слід один з безлічі визначених в мові C ++ операторів.

Мова програмування С++ підтримує шаблони. Шаблони функцій - це узагальнений опис поведінки функцій, які можуть бути викликані об'єктів різних типів. Іншими словами, шаблон функції (шаблонна функція, узагальнена функція) являє собою сімейство різних функцій (або опис алгоритму). За описом шаблон функції схожий на звичайну функцію: різниця в тому, що деякі елементи не визначені (типи, константи) і є параметризовані.

Шаблони класів - узагальнений опис користувальницького типу, в якому можуть бути параметризовані атрибути і операції типу. Являють собою конструкції, за якими можуть бути згенеровані дійсні класи шляхом підстановки замість параметрів конкретних аргументів.

1. Завдання курсової роботи

Побудувати на мові C++ ієрархічну модель класів, що відображає модель реальних об’єктів. Визначити поля і методи об’єктів розроблюваного похідного класу та всіх прямих та опосередкованих базових класів. В залежності від конкретної задачі при побудові ієрархічної моделі використовувати як успадковування, так й агрегацію. Крім того в програмі мають бути використані наступні елементи мови C++.

Внутрішні поля класів повинні бути закритими для зовнішнього доступу.

Доступ до внутрішніх полів має здійснюватися за допомогою відкритих методів доступу та змінення.

В програмі слід визначити перевантажені операції, які б виконували реальні корисні функції для об’єктів розроблюваної моделі.

Для числової статистичної обробки даних різних типів (обчислення середніх значень) необхідно визначити шаблони функцій і застосувати їх.

Отримані масиви об’єктів останнього похідного класу слід зберегти у файлах двійкового формату, і організувати зчитування потрібного об’єкту зі збереженого файлу даних.

У випадках виникнення виключних ситуацій застосувати обробку таких ситуацій.

Програмний продукт для розроблюваної об’єктної моделі повинен мати меню, в якому були б представлені наступні команди:

введення даних;

виведення даних на екран в табличному вигляді (перший рядок - заголовок таблиці, наступні рядки - лише найнеобхідніші дані про кожного об’єкта в окремому рядку);

запис даних на диск у вигляді текстового та окремо бінарного файлу;

зчитування даних з файлів (ім’я файлу повинно пропонуватися програмою);

пошук даних в масиві об’єктів розроблюваного класу за текстовим та числовим полем об’єктів.

За допомогою створеної програми заповнити масив 10 конкретними реальними об’єктами та записати його на диск.

В пояснювальній записці до курсової роботи слід обґрунтувати наступні елементи побудованої моделі:

вибір взаємозв’язків між класами побудованої моделі;

вибір конкретних обмежень на значення внутрішніх полів класів;

вибір необхідних для побудованої моделі перевантажених операцій;

вибір позицій програмного коду для обробки виключних ситуацій;

обґрунтування вибору параметрів типів і параметрів-констант для визначення шаблонів функцій, призначених для підрахунку статистичних функцій, необхідних для побудованої моделі.

У якості варіанту завдання було запропоновано розробити наступну ієрархію класів:

Дата {Місяць, День};

Товар {Назва, Тип (Ваговий, розпакований), Ціна} ;

Асортимент товарів {Список товарів};

Продавець {Прізвище, Ім’я, Вік};

Операція Продажу {Дата (продажу), Продавець, Товар, Кількість товару, Сума операції};

Денний баланс {Список операцій, Загальна денна сума};

Тижневий баланс{Список денних балансів, Загальна тижнева сума}.

Середнє значення кількості товарів проданих за тиждень та середнє значення сум операцій за тиждень.

1. Ієрархія класів

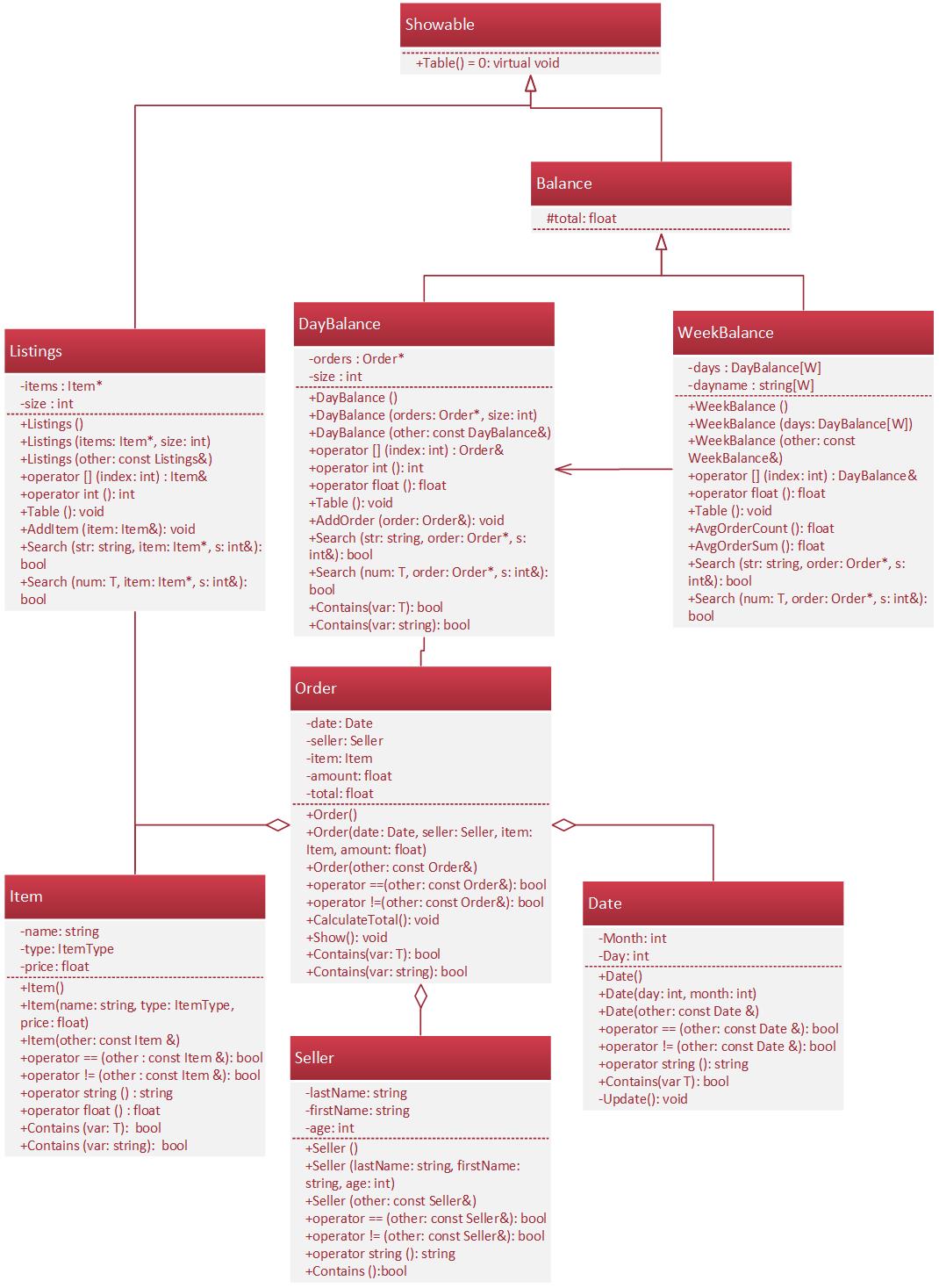
****

Рис.3.1

**Товар** (class Item)

Клас має поле типу товару ItemType type. Воно визначене як тип даних Перерахування. Перерахування (або «перерахований тип») - це тип даних, де будь-яке значення (або «Нумератор») визначається як символьна константа. Оголосити перерахування можна за допомогою ключового слова enum.

Зазвичай перерахування визначається в окремому заголовковому файлі, в даній програмі це файл ItemType.h. Окрім самого перерахування у цьому файлі міститься оголошення методу inline const std::string ToString(ItemType v), який використовується в програмі для зручного виведення типу товару.

enum ItemType { byWeight, individually };

inline const std::string ToString(ItemType v){

switch (v){

case byWeight: return "By weight";

case individually: return "Individually";

default: return "[Unknown]";

}

}

**Операція продажу** (class Order)

У класі представлена агрегація – включені класи Date date, Seller seller, Item item.

Агрегація - це процес об’эднання елементів в одне ціле. У програмуванні це реалізовується за допомогою включення класів у інший клас. Варто зазначити, що при агрегації включені об’єкти можуть належати одночасно кільком класам.

У даному випадку логічно було застосувати саме таку ієрархічну модель тому, що Дата, Продавець та Товар можуть існувати окремо від Операції продажу. Виконується принцип has-a (має).

**Клас** **Showable** (з англ. – той що можна показувати)

Від цього ласу наслідується два класи даної програми: Асортимент товарів та Баланс. Так як від класу Баланс в свою чергу наслідується Тижневий баланс та Денний баланс то **Showable** є батьківським і для них також. У класі оголошений чисто віртуальний метод **virtual void Table() = 0**. Така ієрархія дозволяє зручно користуватися перевизначеними методами для виведення таблиці.

У функцію **void ShowTable(Listings\* listings, WeekBalance\* weekBalance, DayBalance\* dayBalances)**, що знаходиться у файлі **main.cpp** та викликається з меню, передаються створені в програмі об’єкти, які наслідуються від цього абстрактного класу, об’являється вказівник на клас **Showable** і, в залежності від вибору користувача, цей вказівник ініціалізується одним із переданих об’єктів.

Таким чином, при виклику функції **Table()** на цьому вказівнику, буде виконуватися метод відповідного класу і виведеться потрібна таблиця.

**Баланс** (class Balance)

Є базовим класом для класів Тижневий баланс та Денний баланс, наслідується від абстрактного **Showable.**

Метою наслідування є уникнення дублікаціє коду та побудова логічної ієрархії класів з дотриманням принципу is-a (є). При наслідуванні похідний клас отримує всі поля та методи базового класу, але він має безпосередній доступ тільки до членів батьківського класу, які були об’явлені з ідентифікаторами public або protected.

Баланс має одне захищене (protected) поле для суми, яке буде унаслідуване його похідними класами. Перевизначення методу Table у ньому немає тому що сам по собі цей клас не має даних, щоб вивести таблицю.

**Денний баланс** (class DayBalance: public Balance)

При створенні класу використана асоціація – у ньому є вказівник типу Order. Так як Денний баланс може мати кілька покупок то цей вказівник ініціалізується динамічним масивом і містить розмір цього масиву, який повинен завжди ініціалізуватися та/або змінюватися, залежно від кількості об’єктів.

**Тижневий баланс** (class WeekBalance : public Balance)

Клас використовує непряму асоціацію – статичний масив. Днів тижня завжди однакова кількість – 7, тому вирішено було користувати не динамічний а статичний масив.

**Асортимент товарів** (class Listings : public Showable)

Цей клас наслідується від абстрактоного класу Showable, тому що для нього доречно виводити дані у вигляді таблиці. Під час проектування класу використана асоціація – відношення між класами об'єктів, яке дозволяє одному примірнику об'єкта викликати інший, щоб виконати дію від його імені.

На відміну від композиції або агрегації, де частина є частиною цілого, в асоціації об'єкти між собою ніяк не пов'язані. Подібно агрегації, перший об'єкт може належати відразу декільком об'єктам одночасно і не керується ними. Однак, на відміну від агрегації, де відносини односпрямовані, в асоціації відносини можуть бути як односпрямованим, так і двонаправленими (коли обидва об'єкти знають про існування один одного).

**Клас** Utils

Також в програмі розроблений допоміжний клас в якому є два методи для виведення графіки в таблиці та шаблон методу для копіювання даних з одного масиву в інший. Усі методи цього класу є статичними, тобто у пам’яті існує лише один їх екземпляр. Для використання статичного методу потрібно вказувати ім’я класу, якому вони належать та оператор розширення доступу (::). Наприклад Utils::copy(arr1, arr2, size).

* 1. Обмеження полів у класах

При створенні об’єктів класів потрібно враховувати, що поля не можуть набувати певних значень, тому що в реальному світі цього не відбувається. Для цього потрібно встановити певні правила в програмі. Методи, що називаються Сеттери (Setters), використовуються з метою ініціалізувати змінні правильними значеннями.

Поля класів зроблені приватними, так як це відповідає одній із парадигм об’єктно орієнтованого програмування – Інкапсуляції.

Інкапсуляція - властивість мови програмування, що дозволяє користувачеві не замислюватися про складність реалізації використовуваного програмного компонента (що у нього всередині?), А взаємодіяти з ним за допомогою наданого інтерфейсу (публічних методів і членів), а також об'єднати і захистити життєво важливі для компонента дані. При цьому користувачеві надається тільки специфікація (інтерфейс) об'єкта.

**Дата** (class Date)

int day;

int month;

У класі існує два поля типу int – день та місяць. День може набувати значень від 1 до 31, але деякі місяці мають менше днів тому був розроблений метод Update, який змінює значення в залежності від поля Місяць.

void Date::Update()

{

if(month == 2) //лютий місяць

{

if(day > 28)

{

day = 28;

}

}

else

{

if(month % 2 == 0)//парні місяці мають 30 днів

{

if(day > 30)

{

day = 30;

}

}

}

}

Цей метод викликається у сеттерах дати та місяця, а також у конструкторі з параметрами.

Поле місяця має обмеження від 1 до 12, так як це кількість місяців у році.

**Товар** (class Order)

std::string name;

ItemType type;

float price;

Вибране обмеження для імені – воно не можу бути пустим та не повинно бути більше ніж 20 символів. Така кількість символів вибрана через те, що для зручності товар краще називати не надто довгим іменем, бо, наприклад, його потрібно виводити на екран.

Поле ціни обмежене тільки позитивними значеннями та мінімальною ціною – 0.2.

**Продавець** (class Seller)

Поле Вік обмежене значенням 18 років так як працювати в основному починають із такого віку, а також значенням 80 років, бо людина зазвичай не працює продавцем у такому віці.

**Операція продажу** (class Order)

Date date;

Seller seller;

Item item;

float amount;

float total;

У цьому класі обмеження потребує поле кількості (amount) – воно має бути більше нуля і відповідно не може бути від’ємним. В реальному світі кількість товару при покупці в магазині не може бути від’ємною або рівною нулю – тоді операція не має сенсу, товар не був придбаний.

До обмежень полів також можна віднести розрахунок поля суми (total). Знначення цього поля підраховується після того як була встановлена кількість та товар.

void Order::CalculateTotal() {

total = (float)item \* amount;

}

Примітка: приведення об’єкта типу Item до типу float дає значення ціни цього товару.

**Асортимент товарів** (class Listings)

Item\* items;

int size;

Поле розміру динамічного масиву установлюється разом із створенням та ініціалізацією цього масиву. Розмір не може бути меншим або дорівнювати нулеві.

**Денний баланс** (class DayBalance)

Order\* orders;

int size;

А також поле суми, яке унаслідувалося від класу Баланс.

Розмір обмежений ненульовими додатніми значеннями. Сума дня обраховується за допомогою функції

void DayBalance::CalculateTotal()

{

for(int i = 0; i < size; i++)

{

total += orders[i].getTotal();

}

}

Примітка: поле суми при створенні об’єкту Баланс відразу ініціалізується нулем у конструкторі за замовчуванням тому перед циклом не потрібно ще раз це робити.

**Тижневий баланс** (class WeekBalance)

Обмеження схожі на ті що були в класі Денний баланс за винятком розміру масиву – тут він зарані визнаяений як константа.

Для підрахунку суми тижневого балансу використовується функція:

void WeekBalance::CalculateTotal()

{

for(int i = 0; i < W; i++)

{

total += days[i].getTotal();

}

}

1. Поведінка об’єктів
   1. Оператори

Для зручності роботи з класами, використовуються перевизначені оператори. Перевантаження операторів дозволяє визначити дії, які буде виконувати оператор. Перевантаження на увазі створення вибору опцій, що містить слово operator і символ потрібного оператора. Функція оператора може бути визначена як член класу, або поза класом.

Загалом оператори можна поділити на такі групи:

* Арифметичні оператори ( + , - , \* , / , % , ++ , -- )
* Реляційні оператори ( == , != )
* Логічні оператори (&&, ||)
* Побітові оператори (&)
* Оператори присвоювання (=)
* інші оператори

Більшість із операторів можна перевантажувати але з деякими в мові C ++ не можна цього зробити. Очевидно, що це зроблено з міркувань безпеки.

* Оператор вибору члена класу ".".
* Оператор разименованія покажчика на член класу ". \*"
* В С ++ відсутній оператор піднесення до степеня (як в Fortran) "\*\*".
* Заборонено визначати свої оператори (можливі проблеми з визначенням пріоритетів).
* Не можна змінювати пріоритети операторів

У данй програмі присутнє перевантаження таких операторів:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Тип | Date | Item | Listings | Seller | Order | Day Balance | Week Balance |
| Арифметичні |  |  |  |  |  |  |  |
| Реляційні | ==, != | ==, != |  | ==, != | ==, != |  |  |
| Логічні |  |  |  |  |  |  |  |
| Вводу та виводу | << | << |  | << |  |  |  |
| Приведення до типу | string | float, string | int | string |  | int, float | float |
| Індексний |  |  | [ ] |  |  | [ ] | [ ] |

Перевантаження арифметичних операторів було визначене як недоречне у даній програмі, тому що вона працює не з математичними об’єктами.

Реляційні оператори підходять для перевантаження тому, що кожен з об’єктів може дорівнювати або не дорівнювати іншому екземпляру класу. Для їх реалізації потрібно порівняти всі їхні поля. Оператори використовуються по ієрархії класів: у класі Операція продажу цей оператор повинен порівняти поля Дати, Продавця та Товару. Для цього застосовуються перевизначені оператори у їхніх класах.

Також, для класу Дата доречно було б перевизначити інші реляційні оператори такі як більше, менше, більше рівне та менше рівне, але в даній програмі вони не так потрібні.

Оператори вводу та виводу, а саме оператор вводу, що використовується для запису в потік, перевантажений з метою зручного виводу на консоль та запису у файл саме текстової інформації про об’єкт (а не сзапис самого об’єкта в файл). Цей оператор використовує перевантажене приведення типу до string. Це зроблено для того, щоб уникнути дублікації коду.

Оператор вводу не є членом класу у даній програмі. Він визначений поза класом і приймає як параметр екземпляр класу.

std::ostream& operator<<(std::ostream& os, const Date& obj)

{

return (os << std::string(obj));

}

**Приведення до типу**

У розробленій програмі для деяких класів присутнє приведення до типів.

* Дата: приведення до строки. Повертається строка, що створена на основі полів класу але у більше зручному та зрозумілому для користувача форматі. Додається крапка між значенням дня та місяця, а також нуль перед числом, якщо у ньому лише одна цифра. Наприклад, об’єкт із значеннями day = 9, month = 12 поверне рядок «09.12»

Ситаксис:

Date::operator std::string() const {

std::string strM = std::to\_string(month);

if(month < 10)

{

strM = "0" + strM;

}

std::string strD = std::to\_string(day);

if(day < 10)

{

strD = "0" + strD;

}

return strD + "." + strM;

}

* Товар: приведення до строки та до чисельного типу з плаваючою точкою. Перевантаження оператору приведення до строки у цьому класі більше просту ніж у класі Дата – воно просто повертає ім’я товару. Приведення до типу float повертає ціну товару. У програмі використовується у методі підрахунку загальної суми у класі Операція продажу.

* Асортимет товарів: приведення до цілечисельного типу. Повертає кількість товарів. Це дозволяє не викликати метод getSize() для отримання розміру динамічного масиву з товарами.
* Продавець: приведення до строки. Операттор складає ім’я та прізвище продавця через пробіл та повертає строку.

Синтаксис:

Seller::operator std::string() const

{

return lastName + " " + firstName;

}

* Денний баланс: приведення до цілочисельного типу дає кількість Операцій продажу, до типу з плаваючою точкою – загальну денну суму. Можна використовувати при підрахунку тижневого балансу замість метода getTotal;
* Тижневий баланс: приведення до типу з плаваючою точкою дає загальну денну суму.

**Індексний оператор**

У розробленій програмі для деяких класів доречно використовувати перевизначений індексни оператор [ ]. Це дозволяє спростити доступ до внутрішніх масивів у класах та є більш зрозумілим для використання.

Клас Асортимент товарів перевизначає індексний оператор. Він дозволяє доступ до об’єкта Товар із масиву, що знаходиться в цьому класі. Таким чином, не потрібно писати довгу конструкцію listings.getOrders()[0] щоб отримати елемент, достатньо написати назву змінної та квадратні дужки з індексом.

Синтаксис:

Item &Listings::operator[](int index)

{

if(index >= 0 && index < size)

{

return items[index];

}

if(index < 0)

{

throw NegIdxException(index);

}

else if(index >= size)

{

throw LargeIdxException(index);

}

else

{

return items[index];

}

}

Можна помітити, що цей оператор може кинути кілька виключень – коли індекс негативний або коли він більше ніж розмір масиву. Детальніше про обробку виключних ситуацій та ієрархію виключень розглянуто в одному розділі.

Денний баланс: індексний оператор – повертає Операцію замовлення з масиву, синтаксис схожий на той що в класі Асортимент товарів. Відбувається виклик виключень.

Тижневий баланас: оператор повертає об’єкт Денний баланс.

* 1. Пошук

Так як у будь-якій системі використовується пошук, то в програмі повинні бути функції які його здійснюють. У даній програмі пошук працює таким чином:

1. void Search(Listings\* listings, WeekBalance\* weekBalance, DayBalance\* dayBalances) - загальна функція, якій передаються основні об’єкти програми у яких може бути виконаний пошук. У цій же функції відбувається вибір пошуку за текстовим полем чи за числовим, та зчитування пошукового запиту.
2. Далі на потрібному об’єкті викликається метод bool WeekBalance::Search(std::string str, DayBalance\* dayBalance, int& size) – якщо вибране текстове поле, або шаблон функції template <typename T, typename std::enable\_if<std::is\_arithmetic<T>::value>::type\* = nullptr> bool Search(T num, DayBalance\* dayBalance, int& size) – якщо вибране числове поле. Так як за своїм алгоритмом обидва метода схожі, то для спрощення припустимо, що вибраний пошук за текстовим полем.
3. Для кожного вкладеного об’єкта у класі запускається метод bool Contains(std::string str), який повертає значення true, якщо виявлено співпадіння.
4. Метод Contains порівнює текстові поля об’єкта з пошуковим запитом, а також якщо у класі є вкладений то для нього теж запускається однойменний метод (у вкладеному класі)
5. Якщо на якомусь етапі пошуку метод Contains повернув правду то відбувається вихід з цього методу та об’єкт записується у масив який був переданий для результату пошуку
6. Метод Search продовжує свою роботу доти, доки не закінчиться масив об’єктів або масив для результату пошуку.

Для того щоб краще зрозуміти алгоритм пошуку, розглянемо конкретний приклад.

Пошук за текстовим полем в об’єкті Тижневий баланс. Очікуваний результат: два об’єкта типу Денний баланс (виведення таблиць для них).

WeekBalance weekBalance = WeekBalance();

Order order = Order(Date(13, 12), Seller("Tanya", "Gryt"), Item("Bread", individually, 23.0));

try{

weekBalance[2].AddOrder(order);

weekBalance[5].AddOrder(order);

} catch (IndexingException &ex) {

ex.PrintMessage(); }

1. Вибір параметрів пошуку за допомогою меню

Menu

0 - Exit

1 - Create new Object

2 - Show table for...

3 - Read Object from file

4 - Write Object to file

5 - Search in...

5

What do you want to search?

N - number S - string

S

Enter your search

Tanya

1. weekBalance->Search(str, daySearch, size) – виклик методу класу Тижневий баланс для пошуку за текстовим полем.

bool WeekBalance::Search(std::string str, DayBalance\* dayBalance, int& size)

{

if(size < 0)

{

throw NegSizeException();

}

bool flag = false;

int iter = 0;

for(int i = 0; i < W && iter < size; i++)

{

if(days[i].Contains(str))

{

flag = true;

dayBalance[iter] = days[i];

iter++;

}

}

size = iter;

return flag;

}

1. Викликається функція на кожному об’єкті типу Денний баланс if(days[i].Contains(str))

Синтаксис методу:

bool DayBalance::Contains(std::string str)

{

bool flag = false;

for(int i = 0; i < size; i++)

{

if(orders[i].Contains(str))

{

flag = true;

i = size;

}

}

return flag;

}

1. Викликається метод if(orders[i].Contains(str)) на кожному об’єкті класу Операція продажу

Синтаксис методу:

bool Order::Contains(std::string str)

{

bool flag = false;

if(seller.Contains(str))

{

flag = true;

}

else if(item.Contains(str))

{

flag = true;

}

return flag;

}

1. Викликається метод Contains або на об’єкті Продавець або на об’єкті Товар. В даному випадку викликався if(seller.Contains(str))

Синтаксис методу:

bool Seller::Contains(std::string str)

{

bool flag = false;

if(str == lastName || str == firstName)

{

flag = true;

}

return flag;

}

Метод повернув правду адже одне із полів співпало із пошуковим запитом.

1. Метод Пошуку продовжує працювати доти, доки не закінчиться місце у масиві для результату. Потім для знайдених днів в циклі викликається метод Table() для виведення таблиць з Операціями покупка. Як можемо побачити – у них дійсно присутнє шукане слово

Found these Objects

┌─────────────────────┬─────────────────────┬─────────────────────┬─────────────────────┐

│ Date              │ Seller            │ Item              │ Sum               │

╞═════════════════════╪═════════════════════╪═════════════════════╪═════════════════════╡

│ 01.01             │ default default   │default       X0.000 │                0.00 │

├─────────────────────┼─────────────────────┼─────────────────────┼─────────────────────┤

│ 13.12             │ Tanya Gryt        │Bread         X1.000 │                23.00│

├─────────────────────┼─────────────────────┼─────────────────────┼─────────────────────┤

│                     │                     │ Total │ 23.00 │

└─────────────────────┴─────────────────────┴─────────────────────┴─────────────────────┘

┌─────────────────────┬─────────────────────┬─────────────────────┬─────────────────────┐

│ Date              │ Seller            │ Item              │ Sum               │

╞═════════════════════╪═════════════════════╪═════════════════════╪═════════════════════╡

│ 01.01             │ default default   │default       X0.000 │                0.00 │

├─────────────────────┼─────────────────────┼─────────────────────┼─────────────────────┤

│ 13.12             │ Tanya Gryt        │Bread         X1.000 │                23.00│

├─────────────────────┼─────────────────────┼─────────────────────┼─────────────────────┤

│                     │                     │ Total │ 23.00 │

└─────────────────────┴─────────────────────┴─────────────────────┴─────────────────────┘

Found 2 objects

**Пошук за числовим полем**

Другий вид пошуку, що можливий у програмі – це пошук за числовим полем. Він працює таким же чином як і пошук за текстовим полем, спускаючись вниз по ієрархії та перевіряючи поля на співпадіння. Але на цей раз порівнювати потрібно поля які мають числовий тип.

Для того, щоб зробити функцію універсальною, доречно використовувати шаблон функції який би приймав значення будь якого числовго типу (int, float, unsigned int, double і т.д.).

Функція може працювати лише з числовими типами даних тобу перед її декларацією вводимо обмеження. Усі наступні функції Contains мають таке ж.

template <typename T, typename std::enable\_if<std::is\_arithmetic<T>::value>::type\* = nullptr>

Синтаксис функції для пошуку за числовим полем у класі Денний баланс:

template <typename T, typename std::enable\_if<std::is\_arithmetic<T>::value>::type\* = nullptr>

bool Search(T num, Order\* order, int& s)

{

if(s < 0){

throw NegSizeException();

}

bool flag = false;

int iter = 0;

for(int i = 0; i < size && iter < s; i++){

if(orders[i].Contains(num))

{

flag = true;

order[iter] = orders[i];

iter++;

}

}

s = iter;

return flag;

}

* 1. Методи

Окрім операторів, пошуку та виведення таблиць, у класах також присутні інші методи які моделюють дії над даними об’єктами у реальному світі.

**Денний баланс:** метод Додати покупку

void DayBalance::AddOrder(Order order)

До денного балансу можна додати ще одну покупку. При цьому зміниться розмір динамічного масиву. Відбудеться обрахунок нової денної суми

**Асортимент товарів:** метод додати Товар

void Listings::AddItem(Item& item)

Працює аналогічно методу Додати покупку. Є необхідним для роботи з класом, тому що в реальному світі асортимент товару може змінюватися. Також можна було б додати метод для видалення Товару, для повного відображення роботи представленої моделі, але в даній програмі в цьому немає потреби.

**Тижневий баланс:** методи підрахунку статистичної характеристики

Для зручного обчислення та уникнення дублікації коду використано шаблон функції average, який приймає масив типу Т та його розмір. Розмір визначено як unsigned int тому що він не може бути негативним

template <typename T, typename std::enable\_if<std::is\_arithmetic<T>::value>::type\* = nullptr>

float average(T\* arr, unsigned int size){

float sum = 0.0;

for(int i = 0; i < size; i++)

{

sum += arr[i];

}

return (sum / (float)size);

}

Ця функція викликається у наступних методах класу:

float AvgOrderCount(); - для підрахунку середньої кількості покупок за день

float AvgOrderSum(); - для підрахунку середньої суми на день

Так як вони оперують з різними типами даних цілочисельним та з плаваючою точкою – всередині викликається шаблон.

1. Виключення, обробка виключних ситуацій

У процесі роботи програми можуть виникати різні помилки. Наприклад, при передачі файлу по мережі обірветься підключення до мережі або будуть введені некоректні і неприпустимі дані, які викличуть падіння програми. Такі помилки ще називаються виключеннями. Якщо виключення оброблено, то при його виникненні програма припиняє свою роботу.

У мові C ++ оператор throw використовується для сигналізування про виникнення виключення або помилки. Сигналізування про те, що сталося виняток, називається генерацією винятку (або «киданням виключення»).

Для використання оператора throw застосовується ключове слово throw, а за ним вказується значення будь-якого типу даних, яке ви хочете задіяти, щоб сигналізувати про помилку. Як правило, цим значенням є код помилки, будьте готовими описати проблему або настроюється клас-виключення.

* 1. Ієрархія виключень

У програмі існує один базовий клас ArgumentException від якого наслідуються всі інші класи виключень, що розроблені. У свою чергу цей клас має базовий клас exception.

Клас має два поля – для основного та додаткового повідомлення про помилку. Основне повідомлення містить тип помилки, а додаткове – уточнення відносно того, де виникла помилка. Такоє він має один віртуальний метод void PrintMessage(), що перевизначається для всіх класів-нащадків. Така побудова дозволяє обробляти виключну ситуацію не прописуючи всі можливі варіанти виключень.

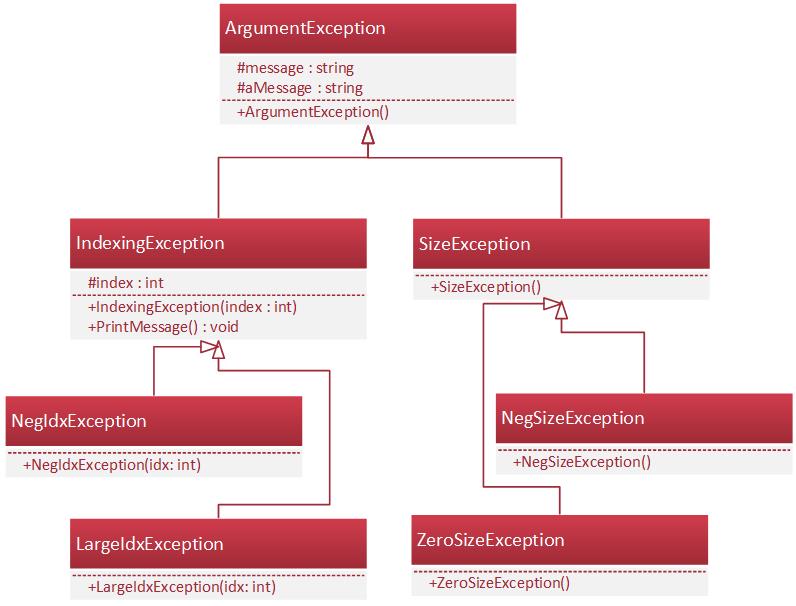


Рис. 5.1

* 1. Обробка виключних ситуацій

**Неправильний розмір**

Може виникнути ситуація коли у метод передається негативне або нульове значення розміру. Для того, щоб програма не ламалася, коли користувач вводить невірне значення для розміру – відбувається обробка.

try{

cout << "How many items do you want to create in listings?" << endl;

int size = -1;

cin >> size;

listings = new Listings(size);

}

catch(SizeException &ex) {

ex.PrintMessage();

}

**Неіснуючий індекс**

У програмі написаній на мові програмування С++ можуть виникати помилки зв’язані з виходом за межі масиву, а таож помилки коли користувач намагається доступитися до елемента з неправильним інндексом – від’ємним.

try{

cout << "What day do you want to edit? from 0 to 6" << endl;

int idx = -1;

cin >> idx;

----

dayBalances[idx].AddOrder(order);

}

catch(NegIdxException &ex)

{

ex.PrintMessage();

}

catch(LargeIdxException &ex){

ex.PrintMessage();

}

**Помилки при роботі з файлами**

При читанні та запису файлу можуть виникати виключні ситуації через те, що файл не вдалося відкрити або у користувача немає доступу до нього. Наявність файлу на диску не залежить від програміста, що пише програму, тому потрібно зробити код надійнішим і поставити обробку виключних ситуацій.

ofstream output;

string filepath = path + name + format;

output.exceptions(ifstream::badbit | ifstream::failbit);

try{

output.open(filepath,ios::out | ios::binary | ios::app);

for(int i = 0; i < size; i++)

{

output.write((char\*)&object[i], sizeof(T));

}

output.close();

}catch (const ifstream::failure &ex){

cout << "Error opening/writing to file" << endl;

cout << "Path: " << filepath << endl;

}

**висновки**

Мова програмування С++ підтримує об’єктно орієнтоване програмування, що робить її зручною для створення моделей об’єктів з реального світу. В ній є можливості використання основних парадигм ООП таких, як інкапсуляція, поліморфізм, успадкування, а також абстрація.

При реалізації наданого завдання були зроблені наступні висновки:

Ієрархічна побудова класів дозволяє зручно та логічно користуватися програмою. Інкапсуляція даних забезпечує їх правильність та надійну взаємодію з ними. Набір методів дає можливість проводити такі дії з об’єктами, які мають місце бути у реальному житті. Для цього самого слугують також перевантажені оператори.

Був проведений аналіз взаємодії користувача з програмою та визначено точки у яких можливе переривання. Для стабільної роботи створено ієрархію виключень, а також застосовано блоки try … catch для обробки виключних ситуацій. Найчастіше такі ситуаціїї виникають при введенні даних та роботі з файлами.

Доступ до полів класів повинен здійснюватися через спеціальні методи з перевірками, що перешкоджає ініціалізації даних об’єкта неправильними значеннями.

Написана програма може бути застосована як частина системи «Магазин» тому, що має ключові її компоненти.

Отже, у ході розробки зазначеної програми закріплено знання з об’єктно-орієнтованого програмування, детальніше розглянуто особливості мови програмування С++ та проаналізовано доцільність її використання при побудові моделей.

**список використаних джерел**

1. Дейтел Х.М., Дейтел П.Дж. Как программировать на Си++. М .: ЗАО БИНОМ, 1999, 1000 с.
2. Лафоре Р. Объектно-ориентированное программирование в С++. Классика Computer Science. 4 изд.. – СПб.: Питер, 2003.
3. Липпман С., Лажойе Ж. Язык программирования C++. Полное руководство, 3-е изд./Пер. с англ. – СПб.: «Невский диалект”, М.: ДМК Пресс. – 1104 с.
4. Павловская Т.А., Щупак Ю.А. С++. Объектно-ориентированное программирование: Практикум. – СПб.: Питер,2004 -288 с.
5. Павловская Т.А.С/С++ Программирование на языке высокого уровня. – СПб.: Питер,2007 -461 с.
6. <https://ravesli.com/urok-134-peregruzka-operatorov-cherez-metody-klassa/>
7. https://www.learncpp.com/cpp-tutorial/103-aggregation/

Додаток 1

Створення моделей об’єктів реального світу на мові C++

Спецификація

Аркушів 2

Київ 2020

|  |  |  |
| --- | --- | --- |
| Позначення | Найменування | Примітки |
| Документація | | |
|  | Записка | Пояснювальна записка |
| Комплекс | | |
|  | Програмний код | Малько\_ПрогрКод.docx |

Додаток 2

Створення моделей об’єктів реального світу на мові C++

Текст програми

Аркушів 19

Київ 2020

Текст програми

#include <iostream>

#include <fstream>

#include "Date.h"

#include "Item.h"

#include "Seller.h"

#include "Order.h"

#include "DayBalance.h"

#include "WeekBalance.h"

#include "Listings.h"

#include "ItemType.h"

#include "Exceptions/IndexingException.h"

#include "Exceptions/LargeIdxException.h"

#include "Exceptions/NegIdxException.h"

#define N 10

using namespace std;

void Create(Listings\* listings, WeekBalance\* weekBalance, DayBalance\* dayBalances);

void ShowTable(Listings\* listings, WeekBalance\* weekBalance, DayBalance\* dayBalances);

void ReadFile(Listings\* listings, WeekBalance\* weekBalance, DayBalance\* dayBalances);

void WriteFile(Listings\* listings, WeekBalance\* weekBalance, DayBalance\* dayBalances);

void Search(Listings\* listings, WeekBalance\* weekBalance, DayBalance\* dayBalances);

template<class T> void WriteTextFile(T\* object, int size, string name = "New file", string path = "", string format = ".txt");

template<class T> void ReadTextFile(T\* object, int size, string name = "New file", string path = "", string format = ".txt");

template<class T> void WriteBinaryFile(T\* object, int size, string name = "New file", string path = "", string format = ".dat");

template<class T> void ReadBinaryFile(T\* object, int size, string name = "New file", string path = "", string format = ".dat");

int main() {

//Заповнення асортимету товарів реальними значеннями

int size = 21;

Listings listings = Listings(size);

listings[0] = Item("White Bread", individually, 12.56);

listings[1] = Item("Dark Bread", individually, 12.56);

listings[2] = Item("Milk 1L", individually, 30.9);

listings[3] = Item("Milk 2L", individually, 30.9);

listings[4] = Item("Butter 67%", byWeight, 200.0);

listings[5] = Item("Butter 75%", byWeight, 250.0);

listings[6] = Item("Butter 82%", byWeight, 300.0);

listings[7] = Item("Rice", byWeight, 30);

listings[8] = Item("Buckwheat", byWeight, 35);

listings[9] = Item("Oats", byWeight, 20.0);

listings[10] = Item("Bulgur", byWeight, 23.5);

listings[11] = Item("Chicken", byWeight, 100.0);

listings[12] = Item("Pork", byWeight, 130.0);

listings[13] = Item("Fish", byWeight, 110.0);

listings[14] = Item("Sausage", byWeight, 110.0);

listings[15] = Item("Dark Chocolate", individually, 25.0);

listings[16] = Item("Milk Chocolate", individually, 27.0);

listings[17] = Item("White Chocolate", individually, 29.0);

listings[18] = Item("Apple Juice", individually, 29.0);

listings[19] = Item("Orange Juice", individually, 29.0);

listings[20] = Item("Cheese", byWeight, 250.0);

WriteTextFile(&listings, 1);

//Заповнення масиву продавців

int sSize = 4;

Seller\* seller = new Seller[sSize];

seller[0] = Seller("Tanya", "Gryt");

seller[1] = Seller("Natalia", "Kuznetsova");

seller[2] = Seller("Daria", "Hrebeniuk");

seller[3] = Seller("Maria", "Volkova");

//Створення масиву з об’єктів класу Операція продажу

Order\* order = new Order[size];

int startDay = 7;

for(int i = 0, j = 0; i < size; i++, j++)

{

//індекс для вибору продавця

if(j == sSize)

{

j = 0;

}

//створення реальних операцій покупок

if(listings[i].getType() == individually)

{

order[i] = Order(Date(startDay++, 12), seller[j], listings[i], (float)(rand() % 8 + 1));

}

else

{

order[i] = Order(Date(startDay++, 12), seller[j], listings[i], (float)((rand() % 2 + 1) + 0.001 \* (rand() % 1000 + 100)));

}

}

//створення тижневого балансу та заповнення його денними балансами

WeekBalance weekBalance = WeekBalance();

int step = 2;

try{

for(int i = 0; i < W; i++)

{

weekBalance[i] = DayBalance(&order[i], step);

}

} catch (IndexingException &ex){

ex.PrintMessage();

}

//додаткове наповнення Денних балансів у Тижневому балансі (для різноманітних значень)

Order order1 = Order(Date(13, 12), seller[0], listings[0]);

try{

order1.setDate(\*weekBalance[2].getOrders()[0].getDate());

weekBalance[2].AddOrder(order1);

order1.setDate(\*weekBalance[5].getOrders()[0].getDate());

order1.setSeller(seller[1]);

weekBalance[5].AddOrder(order1);

} catch (IndexingException &ex)

{

ex.PrintMessage();

}

DayBalance\* dayBalances = new DayBalance[N]; for(int i = 0; i < N; i++)

{

dayBalances[i] = DayBalance(&order[i \* step], step);

}

//зациклене меню

while(true)

{

cout << "Menu" << endl;

cout << "0 - Exit" << endl; //ВИХІД

cout << "1 - Create new Object" << endl; //СТВОРЕННЯ НОВОГО ОБ’ЄКТА

cout << "2 - Show table for..." << endl; //ТАБЛИЦІ

cout << "3 - Read Object from file" << endl; //ЧИТАННЯ З ФАЙЛУ

cout << "4 - Write Object to file" << endl; //ЗАПИС У ФАЙЛ

cout << "5 - Search in..." << endl; //ПОШУК

cout << "6 - Shop stats" << endl; //СТАТИСТИКА ЗА ТИЖДЕНЬ

char t;

cin >> t;

switch (t) {

case '0':

return 0;

case '1':

Create(&listings, &weekBalance, dayBalances);

break;

case '2':

ShowTable(&listings, &weekBalance, dayBalances);

break;

case '3':

ReadFile(&listings, &weekBalance, dayBalances);

break;

case '4':

WriteFile(&listings, &weekBalance, dayBalances);

break;

case '5':

Search(&listings, &weekBalance, dayBalances);

break;

case '6':

cout << "Average order count per day:\t";

cout << weekBalance.AvgOrderCount() << endl;

cout << "Average order sum per day:\t";

cout << weekBalance.AvgOrderSum() << endl;

break;

default:

cout << "Invalid input" << endl;

}

system("pause");

}

}

void Create(Listings\* listings, WeekBalance\* weekBalance, DayBalance\* dayBalances)

{

char o, t;

cout << "Where do you want to store your object?" << endl;

cout << "0 - quit creating\n"

<< "1 - Add Item to listings\n"

<< "2 - Add Order to DayBalance in Week balance\n"

<< "3 - Add Order to DayBalance in array\n";

cin >> o;

string str = "";

string str1 = "";

ItemType itemType = byWeight;

float fValue = 0.0;

char ch;

int size = 1;

//Створення допоміжних локальних об’єктів

Item\* item = new Item[size];

Order\* order;

Seller\* seller;

switch(o)

{

case '0':

return;

case '1':

cout << "Enter the name" << endl;

cin >> str;

cout << "Enter the type: i - sold individually, b - by weight" << endl;

cin >> ch;

if(ch == 'i')

{

itemType = individually;

}

cout << "Enter the price" << endl;

cin >> fValue;

item = new Item(str, itemType, fValue);

listings->AddItem(\*item);

break;

case '2':

try{

cout << "To which day do you want to add order? from 0 to 6" << endl;

int idx = -1;

cin >> idx;

cout << "Enter the name of an Item" << endl;

cin >> str;

size = 1;

listings->Search(str, item, size);

if(size == 0)

{

cout << "Item not found" << endl;

return;

}

cout << "Enter the first name" << endl;

cin >> str;

cout << "Enter the last name" << endl;

cin >> str1;

seller = new Seller(str1, str);

cout << "Enter the amount" << endl;

cin >> fValue;

order = new Order(Date(14, 12), \*seller, \*item, fValue);

weekBalance->operator[](idx).AddOrder(\*order);

}

catch(NegIdxException &ex)

{

ex.PrintMessage();

}

catch(LargeIdxException &ex)

{

ex.PrintMessage();

}

break;

case '3':

try{

cout << "To which day do you want to add order? from 0 to 10" << endl;

int idx = -1;

cin >> idx;

cout << "Enter the name of an Item" << endl;

cin >> str;

size = 1;

listings->Search(str, item, size);

if(size == 0)

{

cout << "Item not found" << endl;

return;

}

cout << "Enter the first name of the seller" << endl;

cin >> str;

cout << "Enter the last name of the seller" << endl;

cin >> str1;

seller = new Seller(str, str1);

cout << "Enter the amount" << endl;

cin >> fValue;

order = new Order(Date(14, 12), \*seller, \*item, fValue);

dayBalances[idx].AddOrder(\*order);

}

catch(IndexingException &ex)

{

ex.PrintMessage();

}

catch(SizeException &ex)

{

ex.PrintMessage();

}

break;

default:

cout << "Invalid input" << endl;

}

}

void ShowTable(Listings\* listings, WeekBalance\* weekBalance, DayBalance\* dayBalances)

{

char o;

int idx;

cout << "What do you want to show?\n";

cout << "0 - quit show\n"

<< "1 - all items in the shop\n"

<< "2 - week balance\n"

<< "3 - day balance (from week balance)\n"

<< "4 - day balance (from array)\n";

cin >> o;

Showable\* spointer; //вказівник на абстраутний клас

switch(o){

case '0':

return;

case '1':

spointer = listings;

break;

case '2':

spointer = weekBalance;

break;

case '3':

cout << "Enter index" << endl;

cin >> idx;

try{

if(idx >= W)

{

throw LargeIdxException(idx);

}

else if(idx < 0)

{

throw NegIdxException(idx);

}

spointer = &(weekBalance->getDays()[idx]);

} catch (IndexingException &ex) {

ex.PrintMessage();

}

break;

case '4':

cout << "Enter index" << endl;

cin >> idx;

try{

if(idx >= N)

{

throw LargeIdxException(idx);

}

else if(idx < 0)

{

throw NegIdxException(idx);

}

spointer = &dayBalances[idx];

} catch (IndexingException &ex) {

ex.PrintMessage();

}

break;

default:

cout << "Unknown command" << endl;

break;

}

spointer->Table();

}

void ReadFile(Listings\* listings, WeekBalance\* weekBalance, DayBalance\* dayBalances)

{

char o, f;

cout << "What file do you want to read?" << endl;

cout << "0 - Text\n1 - Binary\n";

cin >> f;

cout << "What do you want to read from file?" << endl;

cout << "0 - quit writing to file\n"

<< "1 - Listings\n"

<< "2 - Week balance\n"

<< "3 - Day balance\n";

cin >> o;

if(f == '0')

{

switch(o)

{

case '0':

return;

case '1':

ReadTextFile(listings, listings->operator int());

break;

case '2':

ReadTextFile(weekBalance, W);

break;

case '3':

ReadTextFile(dayBalances, N);

break;

default:

cout << "Invalid input" << endl;

}

}

else if(f == '1')

{

switch(o)

{

case '0':

return;

case '1':

ReadBinaryFile(listings, listings->operator int());

break;

case '2':

ReadBinaryFile(weekBalance, W);

break;

case '3':

ReadBinaryFile(dayBalances, N);

break;

default:

cout << "Invalid input" << endl;

}

}

else

{

cout << "Invalid input" << endl;

}

}

void WriteFile(Listings\* listings, WeekBalance\* weekBalance, DayBalance\* dayBalances)

{

char o, f;

cout << "What file do you want to write?" << endl;

cout << "0 - Text\n1 - Binary\n";

cin >> f;

cout << "What do you want to write to file?" << endl;

cout << "0 - quit writing to file\n"

<< "1 - Listings\n"

<< "2 - Week balance\n"

<< "3 - Day balance (from array)\n";

cin >> o;

if(f == '0')

{

switch(o)

{

case '0':

return;

case '1':

WriteTextFile(listings, listings->operator int());

break;

case '2':

WriteTextFile(weekBalance, W);

break;

case '3':

WriteTextFile(dayBalances, N);

break;

default:

cout << "Invalid input" << endl;

}

}

else if(f == '1')

{

switch(o)

{

case '0':

return;

case '1':

WriteBinaryFile(listings, listings->operator int());

break;

case '2':

WriteBinaryFile(weekBalance, W);

break;

case '3':

WriteBinaryFile(dayBalances, N);

break;

default:

cout << "Invalid input" << endl;

}

}

else

{

cout << "Invalid input" << endl;

}

}

void Search(Listings\* listings, WeekBalance\* weekBalance, DayBalance\* dayBalances)

{

char o;

int num;

string str;

char mode;

cout << "What do you want to search?" << endl;

cout << "N - number\tS - string" << endl;

cin >> mode;

cout << "Enter your search" << endl;

if(mode == 'N')

{

cin >> num;

}

else if(mode == 'S')

{

cin >> str;

}

else

{

mode = 'N';

cin >> num;

}

//масиви для запису результатів пошуку

int size = 5; //розмір оновлюється в процесі пошуку

Item\* itemSearch = new Item[size];

DayBalance\* daySearch = new DayBalance[size];

Order\* orderSearch = new Order[size];

cout << "What do you want to search?\n";

cout << "0 - quit show\n"

<< "1 - item in the shop\n"

<< "2 - day balance in the past week\n"

<< "3 - day balance (from array)\n";

cin >> o;

switch(o){

case '0':

break;

case '1':

if(mode == 'N')

{

listings->Search(num, itemSearch, size);

}

else

{

listings->Search(str, itemSearch, size);

}

cout << "Found " << size << " items:" << endl;

for(int i = 0; i < size; i++)

{

cout << itemSearch[i] << endl;

}

break;

case '2':

if(mode == 'N')

{

weekBalance->Search(num, daySearch, size);

}

else if(mode == 'S')

{

weekBalance->Search(str, daySearch, size);

}

cout << "Found " << size << " days:" << endl;

for(int i = 0; i < size; i++)

{

daySearch[i].Table();

}

break;

case '3':

for(int i = 0; i < N; i++)

{

if(mode == 'N')

{

if(dayBalances->Search(num, orderSearch, size))

{

i = N;

}

}

else if(mode == 'S')

{

if(dayBalances->Search(str, orderSearch, size))

{

i = N;

}

}

}

cout << "Found " << size << " orders:" << endl;

for(int i = 0; i < size; i++)

{

orderSearch[i].Show();

}

break;

default:

cout << "Unknown command" << endl;

break;

}

}

//шаблон функції для запису в файл

template<class T> void WriteTextFile(T\* object, int size, string name, string path, string format)

{

ofstream output;

string filepath = path + name + format;

output.exceptions(ifstream::badbit | ifstream::failbit);

try{

output.open(filepath);

for(int i = 0; i < size; i++)

{

output.write((char\*)&object[i], sizeof(T));

}

output.close();

}

catch (const ifstream::failure &ex)

{

cout << "Error opening/writing to file" << endl;

cout << "Path: " << filepath << endl;

}

}

template<class T> void ReadTextFile(T\* object, int size, string name, string path, string format)

{

ifstream input;

string filepath = path + name + format;

input.exceptions(ifstream::badbit | ifstream::failbit);

try{

input.open(filepath);

for(int i = 0; i < size; i++)

{

input.read((char\*)&object[i], sizeof(T));

}

input.close();

}

catch (const ifstream::failure &ex)

{

cout << "Error opening/reading from file" << endl;

cout << "Path: " << filepath << endl;

ex.what();

}

}

template<class T> void WriteBinaryFile(T\* object, int size, string name, string path, string format)

{

ofstream output;

string filepath = path + name + format;

output.exceptions(ifstream::badbit | ifstream::failbit);

try

{

output.open(filepath,ios::out | ios::binary | ios::app);

for(int i = 0; i < size; i++)

{

output.write((char\*)&object[i], sizeof(T));

}

output.close();

}

catch (const ifstream::failure &ex)

{

cout << "Error opening/writing to file" << endl;

cout << "Path: " << filepath << endl;

}

}

//шаблон функції для читання файлу

template<class T> void ReadBinaryFile(T\* object, int size, string name, string path, string format)

{

ifstream input;

string filepath = path + name + format;

input.exceptions(ifstream::badbit | ifstream::failbit);

try{

input.open(filepath, ios::in | ios::binary);

for(int i = 0; i < size; i++)

{

input.read((char\*)&object[i], sizeof(T));

}

input.close();

}

catch (const ifstream::failure &ex)

{

cout << "Error opening/reading from file" << endl;

cout << "Path: " << filepath << endl;

}

}

Додаток 1

Створення моделей об’єктів реального світу на мові C++

Опис програми

Аркушів 9

Київ 2020

Опис програми

Спочатку у програмі створюються три основні об’єкти з якими потім ведеться робота – Асортимент товарів, Тижневий баланс, масив Денних балансів розміром 10.

Асортимент товарів заповняється реальними значеннями продуктів, які продаються у магазині. Далі, створюється масив продавців для зручної ініціалізації покупок, а потім і самі покупки. У кожної ініціалізується товар та продавець, циклічним способом та вибирається випадкова кількість придбаного товару межі якої залежать від типу товару (поштучно чи ваговий).

Створюється один екземпляр тижневого балансу та заповнюється об’єктами денного балансу, які в свою чергу ініціалізуються через конструктор створеними раніше Операціми продажу (покупками). Кожен день має по дві покупки. До деяких денних балансів додається ще одна покупка – для урізноманітнення значень

Потім масив денних балансів ініціалізується схожим чином, який далі записується до файлу.

У програмі наявне меню в якому можна вибрати такі опції:

1. вихід з програми
2. створення нового об’єкту
3. виведення таблиць
4. читання з файлу
5. запис до файлу
6. пошук
7. виведення статистики за тиждень

Створення нового об’єкту

Цей пункт викликає функцію Create та передає в неї основні об’єкти програми (асортимент товарів, тижневий баланс, масив денних балансів). У ній користувач може вибрати три дії: створити новий товар та додати його до асортименту, створити та додати Операцію продажу до певного дня у тижневому балансі або до масиву Денних балансів.

При виборі першого будь якого пункту програма просить заповнити поля для відповідного об’єкта а також запитує у користувача до якого дня потрібно додати Операцію продажу в масиві чи денному балансі.

Виведення таблиць

У цьому пункті користувач має можливість вибрати для якого об’єкта хоче побачити таблицю. Так як основні об’єкти наслідуються від абстрактного класу Showable, то створюється один вказівник на нього і, в залежності від вибору користувача цьому вказівнику надається значення. У разі вибору Денного балансу з Тижневого балансу чи масиву – потрібно ввести індекс бажаного об’єкта.

Читання з файлу

Користувач вибирає з якого формату файлу (текстовий чи бінарний) хоче прочитати та який саме об’єкт.

Запис до файлу

Користувач вибирає до якого формату файлу (текстовий чи бінарний) хоче записати та який саме об’єкт.

5. Пошук

Відбувається пошук у об’єктах які мають метод Search – Listings, WeekBalance, DayBalance. Ці методи приймають пошуковий запит, вказівник (це також може бути масив) на об’єкт куди потрібно записати результат пошуку та розмір масиву. В процесі пошуку розмір змінюється відповідно до кількості знайдених елементів (зменшужться, але не може збільшуватися). Після пошуку знайдені об’єкти виведуться на екран за допомогою своїх методів виведення.

Можна знайти:

Товар у Асортименті товарів

Денний баланс, що містить пошуковий запит - у Тижневому балансі

Операцію продажу – пошук в масиві денних балансів

* 1. Статистика

Виведення на екран статистики за за тиждень – середньої кількості замовлень на день та середньої суми за тиждень. Використовуються методи класу Тижневий баланс.

cout << "Average order count per day:\t";

cout << weekBalance.AvgOrderCount() << endl;

cout << "Average order sum per day:\t";

cout << weekBalance.AvgOrderSum() << endl;

**Демонстрація роботи програми**

Вхідні дані: ім’я товару Yogurt (йогурт), тип – поштучно, ціна 40

Menu

0 - Exit

1 - Create new Object

2 - Show table for...

3 - Read Object from file

4 - Write Object to file

5 - Search in...

6 - Shop stats

1

Where do you want to store your object?

0 - quit creating

1 - Add Item to listings

2 - Add Order to DayBalance in Week balance

3 - Add Order to DayBalance in array

1

Enter the name

Yogurt

Enter the type: i - sold individually, b - by weight

i

Enter the price

40

Press any key to continue . . .

Menu

0 - Exit

1 - Create new Object

2 - Show table for...

3 - Read Object from file

4 - Write Object to file

5 - Search in...

6 - Shop stats

2

2

What do you want to show?

0 - quit show

1 - all items in the shop

2 - week balance

3 - day balance (from week balance)

4 - day balance (from array)

1

1

┌─────────────────────┬─────────────────────┬─────────────────────┐

│ Name              │ Type              │ Price             │

╞═════════════════════╪═════════════════════╪═════════════════════╡

│ White Bread       │ Individually      │ 12.560           │

├─────────────────────┼─────────────────────┼─────────────────────┤

│ Dark Bread        │ Individually      │ 12.560           │

├─────────────────────┼─────────────────────┼─────────────────────┤

│ Milk 1L           │ Individually      │ 30.900           │

├─────────────────────┼─────────────────────┼─────────────────────┤

│ Milk 2L           │ Individually      │ 30.900           │

├─────────────────────┼─────────────────────┼─────────────────────┤

│ Butter 67%        │ By weight         │ 200.000           │

├─────────────────────┼─────────────────────┼─────────────────────┤

│ Butter 75%        │ By weight         │ 250.000           │

├─────────────────────┼─────────────────────┼─────────────────────┤

│ Butter 82%        │ By weight         │ 300.000           │

├─────────────────────┼─────────────────────┼─────────────────────┤

│ Rice              │ By weight         │ 30.000           │

├─────────────────────┼─────────────────────┼─────────────────────┤

│ Buckwheat         │ By weight         │ 35.000           │

├─────────────────────┼─────────────────────┼─────────────────────┤

│ Oats              │ By weight         │ 20.000           │

├─────────────────────┼─────────────────────┼─────────────────────┤

│ Bulgur            │ By weight         │ 23.500           │

├─────────────────────┼─────────────────────┼─────────────────────┤

│ Chicken           │ By weight         │ 100.000           │

├─────────────────────┼─────────────────────┼─────────────────────┤

│ Pork              │ By weight         │ 130.000           │

├─────────────────────┼─────────────────────┼─────────────────────┤

│ Fish              │ By weight         │ 110.000           │

├─────────────────────┼─────────────────────┼─────────────────────┤

│ Sausage           │ By weight         │ 110.000           │

├─────────────────────┼─────────────────────┼─────────────────────┤

│ Dark Chocolate    │ Individually      │ 25.000           │

├─────────────────────┼─────────────────────┼─────────────────────┤

│ Milk Chocolate    │ Individually      │ 27.000           │

├─────────────────────┼─────────────────────┼─────────────────────┤

│ White Chocolate   │ Individually      │ 29.000           │

├─────────────────────┼─────────────────────┼─────────────────────┤

│ Apple Juice       │ Individually      │ 29.000           │

├─────────────────────┼─────────────────────┼─────────────────────┤

│ Orange Juice      │ Individually      │ 29.000           │

├─────────────────────┼─────────────────────┼─────────────────────┤

│ Cheese            │ By weight         │ 250.000           │

├─────────────────────┼─────────────────────┼─────────────────────┤

│ Yogurt            │ Individually      │ 40.000           │

└─────────────────────┴─────────────────────┴─────────────────────┘

Результат: товар був доданий до Асортименту, що видно із таблиці

Запис до файлу

Menu

0 - Exit

1 - Create new Object

2 - Show table for...

3 - Read Object from file

4 - Write Object to file

5 - Search in...

6 - Shop stats

4

4

What file do you want to write?

0 - Text

1 - Binary

1

1

What do you want to write to file?

0 - quit writing to file

1 - Listings

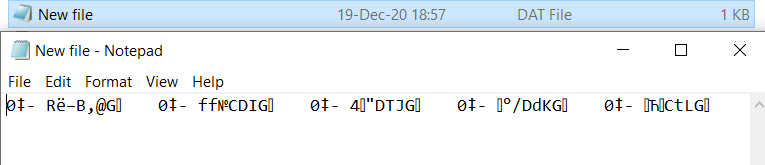
2 - Week balance

3 - Day balance (from array)

3

3

Press any key to continue . . .



Зчитування даних з файлу

Menu

0 - Exit

1 - Create new Object

2 - Show table for...

3 - Read Object from file

4 - Write Object to file

5 - Search in...

6 - Shop stats

3

What file do you want to read?

0 - Text

1 - Binary

1

What do you want to read from file?

0 - quit writing to file

1 - Listings

2 - Week balance

3 - Day balance

3

Press any key to continue . . .

Menu

0 - Exit

1 - Create new Object

2 - Show table for...

3 - Read Object from file

4 - Write Object to file

5 - Search in...

6 - Shop stats

2

What do you want to show?

0 - quit show

1 - all items in the shop

2 - week balance

3 - day balance (from week balance)

4 - day balance (from array)

4

Enter index

0

┌─────────────────────────┬─────────────────────────┬─────────────────────────┬─────────────────────────┐

│ Date                  │ Seller                │ Item                  │ Sum                   │

╞═════════════════════════╪═════════════════════════╪═════════════════════════╪═════════════════════════╡

│ 07.12                 │ Tanya Gryt            │White Bread     X2.000 │                  25.12 │

├─────────────────────────┼─────────────────────────┼─────────────────────────┼─────────────────────────┤

│ 07.12                 │ Natalia Kuznetsova    │Dark Bread      X4.000 │                  50.24 │

├─────────────────────────┼─────────────────────────┼─────────────────────────┼─────────────────────────┤

│                         │                         │ Total │ 75.36 │

└─────────────────────────┴─────────────────────────┴─────────────────────────┴─────────────────────────┘

Press any key to continue . . .

Пошук за текстовим полем

Menu

0 - Exit

1 - Create new Object

2 - Show table for...

3 - Read Object from file

4 - Write Object to file

5 - Search in...

6 - Shop stats

5

What do you want to search?

N - number S - string

S

Enter your search

Milk

What do you want to search?

0 - quit show

1 - item in the shop

2 - day balance in the past week

3 - day balance (from array)

1

Found 3 items:

Milk 1L 30.9

Milk 2L 30.9

Milk Chocolate 27

Пошук за числовим полем

Menu

0 - Exit

1 - Create new Object

2 - Show table for...

3 - Read Object from file

4 - Write Object to file

5 - Search in...

6 - Shop stats

5

5

What do you want to search?

N - number S - string

N

N

Enter your search

7

7

What do you want to search?

0 - quit show

1 - item in the shop

2 - day balance in the past week

3 - day balance (from array)

3

3

Found 2 orders:

┌──────────────────────────┬──────────────────────────┬──────────────────────────┬──────────────────────────┐

│ Date                   │ Seller                 │ Item                   │ Sum                    │

╞══════════════════════════╪══════════════════════════╪══════════════════════════╪══════════════════════════╡

│ 07.12                  │ Tanya Gryt             │White Bread        X2.000 │                     25.12│

└──────────────────────────┴──────────────────────────┴──────────────────────────┴──────────────────────────┘

┌──────────────────────────┬──────────────────────────┬──────────────────────────┬──────────────────────────┐

│ Date                   │ Seller                 │ Item                   │ Sum                    │

╞══════════════════════════╪══════════════════════════╪══════════════════════════╪══════════════════════════╡

│ 07.12                  │ Natalia Kuznetsova     │Dark Bread         X4.000 │                     50.24│

└──────────────────────────┴──────────────────────────┴──────────────────────────┴──────────────────────────┘

Статистика

Menu

0 - Exit

1 - Create new Object

2 - Show table for...

3 - Read Object from file

4 - Write Object to file

5 - Search in...

6 - Shop stats

6

6

Average order count per day: 2.28571

Average order sum per day: 501.467

Press any key to continue . . .